Estructura de Datos y Algoritmos

Grados de la Facultad de Informática de la UCM (Grupos C y F). Curso 2016-2017
Examen parcial de junio Tiempo: 3 horas

Ejercicio 1 [2 puntos]

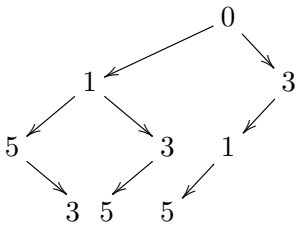
Sean dos listas de números enteros (de tipo `List<int>`) *ordenadas estrictamente de manera creciente*. Implementa una función `diferencia` que modifique la primera lista eliminando de ella todos aquellos elementos que estén presentes en la segunda, manteniendo el orden original. Se valorará la complejidad en tiempo y en espacio de la solución obtenida.

Entrada	Salida
1 2 3 4 -1 3 4 -1	1 2
2 4 6 7 8 9 11 -1 2 11 -1	4 6 7 8 9
1 2 -1 1 2 3 -1	

Ejercicio 2 [2.5 puntos]

Se está poniendo de moda en los casinos un nuevo juego de cartas llamado *WhiteJacket* que consiste en lo siguiente: Juega solo un jugador y hay dos barajas de cartas en la mesa. En cada jugada hay tres opciones, coger una carta del montón izquierdo, coger una carta del montón derecho o plantarse. Se empieza con 0 puntos. Coger una nueva carta (de cualquiera de los montones) resta n puntos, siendo n el número de jugada (la primera jugada resta un punto, la segunda resta dos puntos, etc.), y también suma p puntos siendo p el valor numérico de la carta (siempre positivo).

Sea un árbol binario que representa todas las posibles jugadas del juego (con dos montones de cartas concretos), con valores enteros en sus nodos representando la carta cogida al moverse en la dirección correspondiente, y un 0 en la raíz. Se pide implementar un algoritmo que encuentre la secuencia de jugadas con mejor puntuación. En caso de empate se debe elegir la secuencia más corta, y en caso de empate en longitud la que se corresponda con la sub-rama de más a la izquierda.



La secuencia debe devolverse como un `List<bool>`, siendo `false` coger del montón izquierdo y `true` coger del derecho. Por ej., en el árbol de la figura la mejor secuencia es $0 \to 0$, es decir, la que coge dos cartas del montón izquierdo y luego se planta, obteniéndose 3 puntos (0 al coger la carta 1 y 3 al coger la carta 5). Las secuencias $0 \to 0 \to 1$ y $0 \to 1 \to 0$ también generan 3 puntos pero son más largas. ***Nota**: Puedes obtener hasta 1.5 ptos si implementas un algoritmo que solamente devuelva la puntuación de la mejor secuencia.*

Ejercicio 3 [5 puntos]

Nos piden diseñar y desarrollar el software para el nuevo sistema de grabaciones del nuevo modelo de SmartTV de la marca Sunny. El sistema gestiona y almacena, por un lado, las grabaciones ya realizadas, permitiendo añadir, eliminar y listar grabaciones; y por otro lado, las grabaciones programadas para ser grabadas en el futuro, permitiendo añadir nuevas y listarlas. En particular, el TAD `SmartTV` debe contar con las siguientes operaciones:

- `grabar(TV,N,F,H,T)`: Añade al sistema `TV` una nueva grabación de nombre `N`, fecha y hora de inicio `F` y `H` resp., y tiempo (en minutos) `T`. Si ya existe una grabación con el mismo nombre la operación no tiene efecto. Si esa misma grabación (con igual nombre, fecha y hora de inicio) se encontraba entre las grabaciones programadas, la grabación programada debe eliminarse.
- `eliminarGrabacion(TV,N)`: Elimina del sistema `TV` la grabación de nombre `N`. Si la grabación no existe la operación no tiene efecto.
- `grabacionesRealizadas(TV,L)`: Obtiene en `L` la lista de grabaciones realizadas, ordenadas por nombre.
- `programar(TV,N,F,H,T)`: Programa en el sistema `TV` una nueva grabación a ser realizada, de nombre `N`, fecha y hora de inicio `F` y `H` resp., y tiempo (en minutos) `T`. Si ya existe en el sistema una grabación ya grabada con ese nombre o si la grabación solapa en tiempo con alguna grabación ya programada se produce un error (uno diferente en cada caso).

- `grabacionesProgramadas(TV,F1,F2,L)`: Obtiene en `L` la lista de las grabaciones programadas entre las fechas `F1` y `F2` (ambas incluidas), ordenadas por fecha y hora de inicio.

Se parte del siguiente diseño:

```
class Grabacion{
  string nombre;
  Fecha fecha;
  Hora hora;
  int duracion;
  ...
}
```

```
typedef int Fecha;

class Hora{
  ...
}
class FechaYHora{
  ...
}
```

```
class SmartTV{
  TreeMap<string,Grabacion> grabadas;
  TreeMap<FechaYHora,Grabacion> programadas;
  ...
}
```

Se pide:

a) **[1 punto]** Indicar y justificar el coste asintótico esperado de todas las operaciones del TAD `SmartTV` con la representación proporcionada.

b) **[1.25 puntos]** Implementar las operaciones `grabar`, `programar` y `grabacionesProgramadas` (más las operaciones auxiliares necesarias) con la representación proporcionada.

c) **[1.25 puntos]** Modificar la representación del TAD `SmartTV` de manera que se mejore el coste de la operación `programar` sin empeorar el coste de ninguna otra operación. *Idea: El nuevo coste debe depender del número máximo de grabaciones en un mismo día.* Indicar y justificar el coste de todas las operaciones e implementar la operación `programar` con la nueva representación.

d) **[0.75 puntos]** Implementar la operación `grabacionesProgramadas` con la representación modificada del apartado anterior.

e) **[0.75 puntos]** Diseñar una representación de manera que la operación `programar` tenga coste logarítmico en el número de grabaciones sin penalizar el coste de ninguna otra operación. Explicar detalladamente cómo se implementarían las operaciones afectadas.

Ejercicio 4 - Preguntas cortas [0.5 puntos]

a) ¿Es posible escribir una función *hash* para el tipo `string` que provoque complejidades lineales en el número de elementos de la tabla para inserciones, búsquedas y eliminaciones en un *HashMap*? Justifica tu respuesta y en caso de ser posible escribe un ejemplo de tal función *hash*.

a) Sea la siguiente función `hash` para el tipo `string`:

```
int hash(const string& s){if (s.size()>0) return s[0]; else return 0;}
```

¿Por qué no se comporta bien para tablas *HashMap* grandes? Justifica tu respuesta y pon un ejemplo.

Instrucciones

- Debéis entregar un fichero .cpp para cada ejercicio, nombrado ejerX.cpp siendo X el número del ejercicio. No hace falta que que hagáis ficheros .h.
- Al principio de cada fichero que entreguéis debe aparecer, en un comentario, vuestro nombre y apellidos, dni y puesto de laboratorio. También debéis incluir unas líneas explicando qué habéis conseguido hacer y qué no.
- Todo lo que no sea código C++ (explicaciones, especificaciones, invariantes, etc.) debe ir en los propios ficheros .cpp en comentarios debidamente indicados.
- Los TADs vistos y las plantillas para poder probar vuestras soluciones se obtienen pulsando en el icono del Escritorio "Publicacion docente ...", después en "Alumno recogida docente", y en el programa que se abre, abriendo en la parte derecha la carpeta EDA-F, arrastrando los ficheros a hlocal (en la izqda).
- La entrega se realiza pulsando en el icono del escritorio "Examenes en Labs ...", y posteriormente, utilizando el programa que se abre, colocando los ficheros a entregar en la carpeta de vuestro puesto (en el lado derecho).